Objectives

• You will learn/review:
  – Subsets of Java and Python...
  – That are appropriate for COS 333...
  – Through example programs
    • Example 1 in Java, Python
    • Example 2 in Java, Python
    • …
Agenda

• Prelim: character and string encodings
• How do I use files?
• How do I use arrays?
• How do I use associative arrays?
• How do I compose variadic functions/methods?
Character Encodings

There are many character encodings.

Examples:

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Fixed/Variable Width</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>Fixed</td>
<td>1</td>
</tr>
<tr>
<td>Latin-1</td>
<td>Fixed</td>
<td>1</td>
</tr>
<tr>
<td>UCS-2</td>
<td>Fixed</td>
<td>2</td>
</tr>
<tr>
<td>UCS-4</td>
<td>Fixed</td>
<td>4</td>
</tr>
<tr>
<td>UTF-8</td>
<td>Variable</td>
<td>1, 2, 3, or 4</td>
</tr>
<tr>
<td>UTF-16</td>
<td>Variable</td>
<td>2 or 4</td>
</tr>
</tbody>
</table>
# Character Encodings

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII</th>
<th>Latin-1</th>
<th>UCS-2</th>
<th>UCS-4</th>
<th>UTF-8</th>
<th>UTF-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>20</td>
<td>20</td>
<td>0020</td>
<td>00000020</td>
<td>20</td>
<td>0020</td>
</tr>
<tr>
<td>!</td>
<td>21</td>
<td>21</td>
<td>0021</td>
<td>00000021</td>
<td>21</td>
<td>0021</td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>30</td>
<td>0030</td>
<td>00000030</td>
<td>30</td>
<td>0030</td>
</tr>
<tr>
<td>A</td>
<td>41</td>
<td>41</td>
<td>0041</td>
<td>00000041</td>
<td>41</td>
<td>0041</td>
</tr>
<tr>
<td>a</td>
<td>61</td>
<td>61</td>
<td>0061</td>
<td>00000061</td>
<td>61</td>
<td>0061</td>
</tr>
<tr>
<td>a with grave</td>
<td>e0</td>
<td>00e0</td>
<td>000000e0</td>
<td>e0</td>
<td>00e0</td>
<td></td>
</tr>
<tr>
<td>Greek small pi</td>
<td>03c0</td>
<td>000003c0</td>
<td>cf80</td>
<td>03c0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double prime</td>
<td>2033</td>
<td>00002033</td>
<td>e280b3</td>
<td>2033</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aegean number 2000</td>
<td>00010123</td>
<td>f09084a3</td>
<td>d800dd23</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Strings in Java

• In Java:
  
  `String myString = "abc";`

  • Object of class `String`
  • Internal encoding is `UTF-16`
  • But best to treat as an ADT
Byte Arrays in Java

• In Java:
  
  ```java
  byte[] myBytes = {0x61, 0x62, 0x63};
  ```

• Array of byte values
• No internal encoding
Strings in Python

• In Python:
  - `myStr = 'abc`
    • Object of class `str`
    • Internal encoding is
      – `Latin-1` if possible, or if not…
      – `UCS-2` if possible, or if not…
      – `UCS-4`
    • But best to treat as an ADT
Byte Arrays in Python

• In Python:
  – `myBytes = b'\x61\x62\x63'`
    • Object of class `bytes`
    • Array of bytes
    • No internal encoding
Agenda

• Prelim: character and string encodings
• **How do I use files?**
• How do I use arrays?
• How do I use associative arrays?
• How do I compose variadic functions/methods?
“CopyBytes” Program

• The job
  – Accept names of input and output files as command-line args
  – Read some bytes from input file; write those bytes to output file; repeat until end of input file
“CopyBytes” in Java

• See CopyBytes.java
  – Command-line arguments
    • main() parameter
  – FileInputStream object
    • read() method
      – Reads bytes from file to array of byte values
  – FileOutputStream object
    • write() method
      – Writes bytes from array of byte values to file
“CopyBytes” in Python

• See `copybytes.py`
  – Command-line arguments
    • `sys.argv` is a global variable
  – `open()` function with mode ‘rb’
    • `read()` function reads bytes from file to `bytes` object
  – `open()` function with mode ‘wb’
    • `write()` function writes bytes from `bytes` object to file
“CopyStrings” Programs

• The job:
  – Accept names of input and output files as command-line args
  – Read a line (as a string) from input file; write the line (as a string) to output file, repeat until end of input file
“CopyStrings” in Java

• See `CopyStrings.java`
  – `BufferedReader` object
    • Wraps around `InputStreamReader` object
      – Wraps around `FileInputStream` object
    • Reads bytes, and decodes to `String`
  – `PrintWriter` object
    • Wraps around `BufferedWriter` object
      – Wraps around `OutputStreamWriter` object
        » Wraps around `FileOutputStream` object
    • Encodes `String` to bytes, and writes
“CopyStrings” in Java

• See CopyStrings.java (cont.)
  – Slower than CopyBytes.java, but…
  – Data can be manipulated as Strings
    • Sorted, concatenated, …

• Generalizing…
Java I/O

A few Java I/O classes (more than 60 of them!!!):

**InputStream**: reads bytes
- **FileInputStream**: source; reads bytes from a file
- **BufferedInputStream**: decorator; adds buffering

**OutputStream**: writes bytes
- **FileOutputStream**: sink; writes bytes to a file
- **BufferedOutputStream**: decorator; adds buffering
- **PrintStream**: decorator; adds C-style formatting

**Reader**: reads Strings; respects character encoding
- **InputStreamReader**: bridge
- **BufferedReader**: decorator; adds buffering, readLine()

**Writer**: writes Strings; respects character encoding
- **OutputStreamWriter**: bridge
- **BufferedWriter**: decorator; adds buffering
- **PrintWriter**: bridge, decorator; adds C-style formatting

All are in package java.io

System.in is of class InputStream
System.out and System.err are of class PrintStream
Java I/O

• **Dimension 1:** InputStream/OutputStream vs. Reader/Writer
  - **InputStream:** reads into bytes array
  - **OutputStream:** writes from bytes array
  - **Reader:** reads and decodes into Strings
  - **Writer:** encodes and writes from Strings
Java I/O

• **Special cases**: `System.out` and `System.err`
  – `OutputStream` objects
    • More specifically, `PrintStream` objects
  – Should be `Writer` objects
  – Use default encoding for particular host
Java I/O

• **Dimension 2:** Source vs. sink vs. bridge vs. decorator
  – **Source:** stream from which data is read
  – **Sink:** stream to which data is written
  – **Bridge:** crosses from stream to reader/writer
  – **Decorator:** adds functionality to stream or reader/writer
Java I/O

To read bytes:

\[
\begin{array}{l}
\text{InputStream object (decorator)} \\
\text{InputStream object (decorator)} \\
\text{InputStream object (source)}
\end{array}
\]

To write bytes:

\[
\begin{array}{l}
\text{OutputStream object (decorator)} \\
\text{OutputStream object (decorator)} \\
\text{OutputStream object (sink)}
\end{array}
\]
Java I/O

To read Strings:

Reader object (decorator)
Reader object (decorator)
Reader object (bridge)
InputStream object (source)

To write Strings:

Writer object (decorator)
Writer object (decorator)
Writer object (bridge)
OutputStream object (sink)
“CopyStrings” in Python

• See **copystrings.py**
  – `open()` function with mode ‘r’
    • `read()` function reads bytes and decodes to `str`
  – `open()` function with mode ‘w’
    • `write()` function encodes `str` to bytes and writes
  – Implicit call of `read()` via `for` statement!
    • `for` statement can iterate over any `Iterable` object – even a file!
    • More on that later
“CopyStrings” in Python

• See `copystrings.py` (cont.)
  – Slower than `copybytes.py`, but…
  – Data can be manipulated as Strings
    • Sorted, concatenated, …
Agenda

• Prelim: character and string encodings
• How do I use files?
• **How do I use arrays?**
• How do I use associative arrays?
• How do I compose variadic functions/methods?
“LineSort” Programs

• The job:
  – Read lines from stdin
  – Sort lines in lexicographical order
  – Write lines to stdout
“LineSort” in Java

• See LineSort1.java
  – Fixed-size array
    • Simple, but unrealistic
  – Reading via Scanner object
  – Sorting via Arrays.sort()
  – Writing via System.out.println()
  – CPU time to sort Bible8.txt: 4.4 sec
“LineSort” in Java

• See LineSort2.java
  – Generic ArrayList class
    • Handles any number of lines
  – Reading via Scanner object
  – Writing via System.out.println()
  – Sorting via Collections.sort()
  – “Foreach” statement
  – CPU time to sort Bible8.txt: 4.2 sec
“LineSort” in Java

• See LineSort3.java
  – Input buffering
  – Output buffering
    • Beware: Must flush buffered output!!!
  – Verbosity!!!
  – CPU time to sort Bible8.txt: 1.9 sec!!!
“LineSort” in Python

- See `linesort1.py`
  - `list`: variable sized heterogeneous array
    - Has `sort()` method
  - Can iterate over a file or list
  - CPU time to sort Bible8.txt: 3.5 sec!!
    - Faster than expected!?
    - Can you guess why???
“LineSort” in Python

• See linesort2.py
  – Uses custom mergesort
  – CPU time to sort Bible8.txt: 7.5 sec
    • Slow!!! Can you guess why???
“Formatter” Programs

• The job:
  – Read from stdin
  – Write to stdout
    • No more than 60 chars per line
    • Don’t split words
• Prof. Kernighan uses to format e-mail
• Keep in mind for Assignment 1
“Formatter” in Java

• See **Formatter.java**
  – Regular expressions
    • Example: "\\S+"
      – A sequence of 1 or more non-white-space characters
“Formatter” in Python

• See `formatter.py`
  – Strings as objects
    • `len(seq)` calls `seq.__len__()`
  – Raw string
    • Example: `r'\S+'`
    • Backslash is not interpreted as an escape character
  – Regular expression
    • Example: `\S+`
      – A sequence of 1 or more non-white-space characters

• See appendix for more info on regular expressions
Agenda

• Prelim: character and string encodings
• How do I use files?
• How do I use arrays?
• How do I use associative arrays?
• How do I compose variadic functions/methods?
“Concordance” Programs

• The job:
  – Read words from stdin
  – Write to stdout
    • Each word and occurrence count
"Concordance" in Java

- See **Concord1.java**
  - Associative array = *HashMap* object
  - Regular expression
    - Defines a pattern
    - Used to select substring(s) of a given string
    - Example: `[a-z]+`
      - A sequence of 1 or more lower-case letters
    - See Appendix for more info
  - Awkwardness of storing *Integer* objects instead of *int* values
  - CPU time to handle Bible8.txt: 1.0 sec
“Concordance” in Java

• See **Concord2.java**
  – Autoboxing
    • `concordance.put(word, 1);`
    • Compiler automatically generates:
      • `concordance.put(word, new Integer(1));`
  – Autounboxing
    • `int count = concordance.get(word);`
    • Compiler automatically generates:
      • `int count = concordance.get(word).intValue();`
“Concordance” in Java

– Both autounboxing and autoboxing!!!
  • count++;
  • Compiler automatically generates:
  • count =
    new Integer(count.intValue() + 1);

– CPU time to handle Bible8.txt: 1.0 sec
Aside: Java Autoboxing

• Autoboxing and autounboxing
  – Necessary because Java is a hybrid lang
  – Bridge the gap between primitive data and objects
  – `Integer I = 123;`  
    // no need for new `Integer()`  
  – `int i = I;`  
    // no need for `I.intValue()`  
  – **Same for** `Long, Float, Double, Short, Byte, Character, Boolean`
Aside: Java Autoboxing

• What does this write (from Josh Bloch via Brian Kernighan)

```java
public class Box {
    public static void main(String[] args) {
        cmp(new Integer(42), new Integer(42));
    }
    private static void cmp(Integer first, Integer second) {
        if (first < second)
            System.out.println("first is less than second");
        else if (first == second)
            System.out.println("first equals second");
        else if (first > second)
            System.out.println("first is greater than second");
    }
}
```
“Concordance” in Java

• Generalizing
  – Java container interfaces and classes...
Java Container Interfaces

Some Java container *interfaces*:

- **Iterable**: an object over which one can iterate
- **Collection**: the root interface
  - **List**: an ordered collection
  - **Set**: a collection that contains no duplicates
    - **SortedSet**: a set that orders its elements
- **Map**: an object that maps keys to values
  - **SortedMap**: a map that orders its keys

All are in package `java.util`
### Java Container Classes

Some Java container **classes**:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ArrayList</strong></td>
<td>a List implemented as an array</td>
</tr>
<tr>
<td><strong>LinkedList</strong></td>
<td>a List implemented as a linked list</td>
</tr>
<tr>
<td><strong>HashSet</strong></td>
<td>a Set implemented as a hash table</td>
</tr>
<tr>
<td><strong>TreeSet</strong></td>
<td>a SortedSet implemented as a red-black tree</td>
</tr>
<tr>
<td><strong>HashMap</strong></td>
<td>a Map implemented as a hash table</td>
</tr>
<tr>
<td><strong>TreeMap</strong></td>
<td>a SortedMap implemented as a red-black tree</td>
</tr>
</tbody>
</table>

**Arrays**: static methods related to arrays  
**Collections**: static methods related to Collection objects

All are in package `java.util`

See *Java Summary* document for more info
“Concordance” in Python

• See **concord.py**
  – Regular expression
    • Example: `[a-z]`+
      – A sequence of 1 or more lower-case letters
  – Raw strings
    • Example: `r' [a-z]+ '`
    • Backslash is not interpreted as an escape character
“Concordance” in Python

- Associative array = `dict` object
  - `in` operator
  - Indexing using `[]` operator
  - Iteration using `for` statement

- CPU time to handle Bible8.txt: 2.8 sec

- Generalizing...
  - Python container classes
Python Iterable Classes

Iterable
  Container
    Mapping
      dict: a hash table, alias associative array
  Sequence
    str: an immutable sequence of characters
    bytes: an immutable sequence of bytes
    tuple: an immutable sequence of objects
    list: a sequence of objects
    set: a group of objects that contains no dups
    file

Note: str and bytes objects are sequences
Note: str, bytes, and tuple objects are immutable
Creating Iterable Objects

```
str
  strobj1 = 'hi'
  strobj2 = "hi"
  strobj3 = r'hi'

bytes
  bytesobj1 = b'hi'
  bytesobj2 = b"hi"
  bytesobj3 = rb'hi'

tuple
  tupleobj1 = (obj1, obj2, ...)
  tupleobj2 = (obj1, ) ← hack

list
  listobj = [obj1, obj2, ...]

set
  setobj = set(listobj) # tests for object ref equality

dict
  dictobj = {keyobj1:valueobj1, keyobj2:valueobj2, ...}

file
  fileobj = open('filename', mode='somemode')
    # somemode: r, rb, w, wb, ...
```

See *Python Summary* document for more info
Agenda

• Prelim: character and string encodings
• How do I use files?
• What array data structures are provided?
• What associative array data structures are provided?
• How do I compose variadic functions/methods?
“Variadic” Programs

• **Variadic** function/method:
  – A function/method that can be called with a variable number of arguments
  – Example: `printf()` in C
    • `printf("hello");`
    • `printf("The answer is %d", 5);`
    • `printf("The answers are %d and %d", 5, 10);`
“Variadic” Programs

• See **Variadic.java**
  – Method accepts a variable number of String arguments
  – Added to Java to handle System.out.printf() and System.err.printf()

• See **variadic.py**
  – Function accepts some arguments by position, some in a list, some in a dictionary
python is a very easy language. i think it's actually a good choice for some things. awk is perfect for a line or two, python for a page or two. both break down badly when used on larger examples, although python users utterly refuse to admit its weaknesses for large-scale programming, both in syntax and efficiency.

-rob

What do you think?
Summary

We have covered the answers to these questions for Java and Python:

– How do I use files?
– How do I use arrays?
– How do I use associative arrays?
– How do I compose variadic functions/methods?
Summary

• We have covered:
  – Subsets of Java and Python...
  – That are appropriate for COS 333...
  – Through example programs
    • Example 1 in Java, Python
    • Example 2 in Java, Python
    • …
Appendix 1: Regular Expressions
Regular Expressions

• Used widely
  – Java (string manipulation)
  – Python (string manipulation)
  – Unix `grep` command (file searching)
  – Bash shell (filename wildcards)
  – SQL `like` clauses (querying databases)
    • See upcoming *Databases* lectures
  – …
# Regular Expression Examples

<table>
<thead>
<tr>
<th>RE</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>thing</code></td>
<td>thing anywhere in string</td>
</tr>
<tr>
<td><code>^thing</code></td>
<td>thing at beginning of string</td>
</tr>
<tr>
<td><code>thing$</code></td>
<td>thing at end of string</td>
</tr>
<tr>
<td><code>^thing$</code></td>
<td>string that contains only thing</td>
</tr>
<tr>
<td><code>^</code></td>
<td>any string, even empty</td>
</tr>
<tr>
<td><code>^$</code></td>
<td>empty string</td>
</tr>
<tr>
<td><code>.</code></td>
<td>non-empty, i.e. the first char in string</td>
</tr>
<tr>
<td><code>thing.$</code></td>
<td>thing plus any char at end of string</td>
</tr>
<tr>
<td><code>thing\.$</code></td>
<td>thing. at end of string</td>
</tr>
<tr>
<td><code>\\thing\</code></td>
<td>\thing\ anywhere in string</td>
</tr>
<tr>
<td><code>[tT]hing</code></td>
<td>thing or Thing anywhere in string</td>
</tr>
<tr>
<td><code>thing[0-9]</code></td>
<td>thing followed by one digit</td>
</tr>
<tr>
<td><code>thing[^0-9]</code></td>
<td>thing followed by a non-digit</td>
</tr>
<tr>
<td><code>thing[0-9][^0-9]</code></td>
<td>thing followed by digit, then non-digit</td>
</tr>
<tr>
<td><code>thing1.*thing2</code></td>
<td>thing1 then any (or no) text then thing2</td>
</tr>
<tr>
<td><code>^thing1.*thing2$</code></td>
<td>thing1 at beginning and thing2 at end</td>
</tr>
</tbody>
</table>

Thanks to Prof. Brian Kernighan
Regular Expression Examples

• What do these match?
  - a.*e.*i.*o.*u
    • Try with `grep` command and `/usr/share/dict/words` file
  - ^[^aeiou]*a[^aeiou]*e[^aeiou]*i[^aeiou]*o[^aeiou]*u[^aeiou]*$
    • Try with `grep` command and `/usr/share/dict/words` file

Thanks to Prof. Brian Kernighan
Regular Exprs in Java/Python

• Implementations vary
  – See *Mastering Regular Expressions* (Jeffrey Friedl) book
    • ~500 pages!

• Common to Java and Python (and `grep`)...
## Regular Expressions in Java/Python

<table>
<thead>
<tr>
<th>RE</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>the character X, except for metacharacters</td>
</tr>
<tr>
<td>\X</td>
<td>the character X, where X is a metacharacter</td>
</tr>
<tr>
<td>.</td>
<td>any character except \n</td>
</tr>
<tr>
<td></td>
<td>(use DOTALL as argument to compile() to match \n too)</td>
</tr>
<tr>
<td>^</td>
<td>start of string</td>
</tr>
<tr>
<td>$</td>
<td>end of string</td>
</tr>
<tr>
<td>XY</td>
<td>X followed by Y</td>
</tr>
<tr>
<td>X*</td>
<td>zero or more cases of X (X*? is the same, but non-greedy)</td>
</tr>
<tr>
<td>X+</td>
<td>one or more cases of X (X+? is the same, but non-greedy)</td>
</tr>
<tr>
<td>X?</td>
<td>zero or one case of X (X?? is the same, but non-greedy)</td>
</tr>
<tr>
<td>[...]</td>
<td>any one of ...</td>
</tr>
<tr>
<td>^...</td>
<td>any character other than ...</td>
</tr>
<tr>
<td>[X-Y]</td>
<td>any character in the range X through Y</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>(...)</td>
<td>... , and indicates a group</td>
</tr>
</tbody>
</table>

Precedence: * + ? higher than concatenation, which is higher than |
## Regular Exprs in Java/Python

**RE** | **Matches**
---|---
\t | tab
\v | vertical tab
\n | newline
\r | return
\f | form feed
\a | alert
\e | escape
\\ | backslash
\A | empty string at start of given string
\b | empty string, but only at start or end of a word
\B | empty string, but not at start or end of a word
\d | a digit
\D | a non-digit
\s | a white space character, that is, \[\t\n\r\f\v\]
\S | a non-white space character
\w | an alphanumeric character, that is, \[a-zA-Z0-9_\]
\W | a non-alphanumeric character
\Z | the empty string at the end of the given string
More Reg Exp Examples

• What kinds of strings do these regular expressions match?
  – /\*.*?\*/ (use with DOTALL)
    • Why the question mark?
    • Why DOTALL?

• Commentary: Regular expressions are write-only!!!
Regular Exprs and DFAs

• Some theory:
  – Regular expressions have the same power as deterministic finite state automata (DFAs)
  – A regular expression defines a regular language
  – A DFA also defines a regular language
Regular Exprs and DFAs

ab*c

\[ a \rightarrow b \rightarrow c \rightarrow \text{accept} \]

a[bc]+d

\[ a \rightarrow b,c \rightarrow d \rightarrow \text{accept} \]
Appendix 2: C Programming
“LineSort” in C

• See linesort1.c
  – Fixed-size arrays
    • Simple, but unrealistic
  – Reading via fgets()
    • Must eliminate trailing '\n'
  – Sorting via qsort()
    • Requires callback compare function
  – Writing via puts()
    • Appends trailing '\n'
  – CPU time consumed
“LineSort” in C

• See linesort2.c
  – Dynamically expanding array
    • Handles any number of lines
  – (Max line length still constant)
  – CPU time consumed
“LineSort” in C

• See **linesort3.c**
  – DynArray object
    • From COS 217
    • Similar to Java ArrayList
  – (Max line length still constant)
  – CPU time consumed

• Generalizing
  – File I/O...
C File I/O

Open/close
FILE *psFile;
psFile = fopen("filename", "mode");
    /* mode: r, w, a, r+, w+, a+ */
fclose(psFile);

Write/read a char
iStatus = fputc(i, psFile);
iStatus = putc(i, psFile);
iStatus = putchar(i);
i = fgetc(psFile);
i = getc(psFile);
i = getchar();
C File I/O

Write/read a string
iStatus = fputs(pcStr, psFile);
iStatus = puts(pcStr);
pcStatus = fgets(pcStr, iBufSize, psFile);
pcStatus = gets(pcStr);

Write/read formatted data
iCharsWritten = fprintf(psFile, "%d", i);
iCharsWritten = printf("%d", i);
iValuesRead = fscanf(psFile, "%d", &i);
iValuesRead = scanf("%d", &i);
“Formatter” in C

• See **formatter.c**
  – Command-line arguments
    • No args => read from stdin
    • One arg => read from that file
    • Otherwise => error
  – File I/O
  – Wrappers around standard functions
  – *Absence of* regular expressions
    • `strtok()` instead
“Concordance” in C

• See `concord.c`
  – Lack of regular expressions
    • Awkward (incorrect!) use of `strtok()`
  – Lack of associative arrays
    • Must create your own
    • Recall `SymTable` module from COS 217
  – CPU time consumed
Appendix 3: Java Debugging
Java Debugging

- jdb debugger is bundled with JDK
- To use jdb:
  - $ javac -g Class.java
  - $ jdb Class
Some \texttt{jdb} Commands

- \texttt{help}
- \texttt{stop in pkg.pkg.Class.method}
- \texttt{stop in pkg.pkg.Class.method(paramtype, paramtype, ...)}
- \texttt{stop at pkg.pkg.Class:line}
- \texttt{clear pkg.pkg.Class.method}
- \texttt{clear pkg.pkg.Class.method(paramtype, paramtype, ...)}
- \texttt{clear pkg.pkg.Class:line}
- \texttt{run}
- \texttt{run pkg.pkg.Class}
- \texttt{run pkg.pkg.Class args}
- \texttt{list}
- \texttt{next}
- \texttt{step}
- \texttt{cont}
- \texttt{print expr}
- \texttt{where}
- \texttt{quit}
Java Debugging

- Type `!!` to repeat the last command
- Beware: `Cannot read from stdin`
Appendix 4: Python Debugging
Python Debugging

• **pdb debugger** is bundled with Python

• **To use pdb:**
  
  ```
  $ python -m pdb file.py
  ```
Python Debugging

Some `pdb` Commands

- help
- break `functionOrMethod`
- break `filename:linenum`
- run
- list
- next
- step
- continue
- print `expr`
- where
- quit
Python Debugging

• Common commands have abbreviations: h, b, r, l, n, s, c, p, w, q
• Blank line means repeat the same command
• Beware: Cannot easily read from stdin
Appendix 5:
C Debugging
C Debugging

- **gdb** debugger from GNU
  - Integrated with **emacs** editor
  - As described in COS 217

- **To use gdb:**
  - `$ gcc -g program.c -o program`
  - `$ emacs`
  - `<Esc>x gdb<Enter>program<Enter>`
C Debugging

Some `gdb` Commands

- `help`
- `break function`
- `break filename:linenum`
- `run`
- `list`
- `next`
- `step`
- `continue`
- `print expr`
- `where`
- `quit`
C Debugging

- Common commands have abbreviations: h, b, r, l, n, s, c, p, q
- Blank line means repeat the same command